Available online at www.sciencedirect.com
SCIENCE DIRECT®
Future Generation Computer Systems 20 (2004) 489–500
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/future

Using the parallel algebraic recursive multilevel solver in modern physical applications

M. Sosonkina [a,*], Y. Saad [b], X. Cai [c]

[a] *Department of Computer Science, University of Minnesota Duluth, 320 Heller Hall, 1114 Kirby Dr., Duluth, MN 55812, USA*
[b] *Department of Computer Science and Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455, USA*
[c] *Simula Research Laboratory and University of Oslo, P.O. Box 1080, Blindern, N-0316 Oslo, Norway*

Abstract

This paper discusses the application of a few parallel preconditioning techniques, which are collected in a recently developed suite of codes Parallel Algebraic Recursive Multilevel Solver (pARMS), to tackling large-scale sparse linear systems arising from real-life applications. In particular, we study the effect of different algorithmic variations and parameter choices on the overall performance of the distributed preconditioners in pARMS by means of numerical experiments related to a few realistic applications. These applications include magnetohydrodynamics, nonlinear acoustic field simulation, and tire design.

Keywords: Parallel algebraic multilevel preconditioning; Distributed sparse linear systems; Nonlinear acoustic field simulation; Magnetohydrodynamic flows; Tire design

1. Introduction

Viable solutions of modern computational problems that arise in science and engineering should efficiently utilize the combined power of multi-processor computer architectures and effective algorithms. For many large-scale applications, solving large sparse linear systems is the most intensive computational task. The important criteria for a suitable solver include numerical efficiency, robustness, and good parallel performance. However, many existing parallel solvers are either designed for a particular problem class, such as symmetric positive definite linear systems, or are very application- and data format-specific. There is a limited selection of "general-purpose" distributed-memory iterative solution implementations. Among the better known packages that contain such implementations are PETSc [1] and *hypre* [6]. While the former focuses mainly on domain decomposition preconditioning techniques, the latter has a wide range of preconditioners including various distributed incomplete LU factorizations and an algebraic multigrid method. In this paper, we consider the Parallel Algebraic Recursive Multilevel Solver (pARMS) [18], which is a suite of distributed-memory iterative accelerators and preconditioners targeting the solution of general sparse linear systems. It adopts a general framework of distributed sparse matrices and relies on the solution of the resulting distributed Schur complement systems. We will discuss some issues related to the performance of pARMS on parallel

* Corresponding author. Tel.: +1-218-726-6149; fax: +1-218-726-8240.
E-mail addresses: masha@d.umn.edu (M. Sosonkina), saad@cs.umn.edu (Y. Saad), xingca@simula.no (X. Cai).
URLs: http://www.d.umn.edu/∼masha, http://www.cs.umn.edu/∼saad, http://www.ifi.uio.no/∼xingca.

0167-739X/$ – see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/S0167-739X(03)00189-4

computers for a few sparse linear systems that arise in realistic applications. Section 2 gives an introduction to the framework of distributed sparse linear systems, whereas Section 3 explains the ARMS method as the underpinning for the preconditioners in pARMS, followed by a discussion of the issues pertinent to their parallel implementations. Section 4 contains numerical experiments arising from three realistic applications. Finally, Section 5 presents some concluding remarks.

2. Distributed sparse linear systems

The framework of distributed linear systems [23,26] provides an algebraic representation of the irregularly structured sparse linear systems arising in the Domain Decomposition methods [30]. A typical distributed system arises, e.g., from a finite element discretization of a partial differential equation on a certain domain. To solve such systems on a distributed memory computer, it is common to partition the finite element mesh and assign a cluster of elements representing a physical sub-domain to one processor. Each processor then assembles only the local equations restricted to its assigned cluster of elements. In the case where the linear system is given algebraically, a graph containing vertices that correspond to the rows of the linear system can be partitioned. For both cases, the general assumption is that each processor holds a set of equations (rows of the global linear system) and the associated unknown variables.

Fig. 1 shows a 'physical domain' viewpoint of a distributed sparse linear system, for which a vertex-based

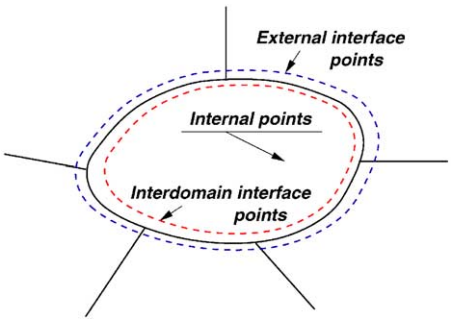


Fig. 1. A local view of a distributed sparse linear system.

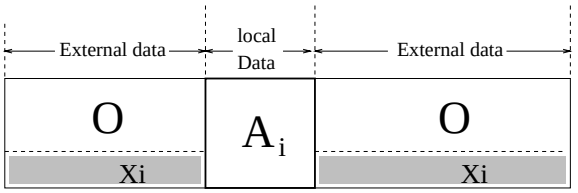


Fig. 2. The corresponding local matrices A_i and X_i.

partitioning has been used without overlapping the unknowns. As is often done, we will distinguish between three types of variables: (1) interior variables are those that are coupled only with local variables by the equations; (2) inter-domain interface variables are those coupled with nonlocal (external) variables as well as local variables; and (3) external interface variables are those variables that belong to neighboring processors and are coupled with the above types of local variables. The local equations can be represented as shown in Fig. 2. Note that these equations need not be contiguous in the original system. The matrix represented in Fig. 2 can be viewed as a reordered version of the equations associated with a local numbering of the equations/unknowns pairs.

The rows of the matrix assigned to a certain processor have been split into two parts: a *local* matrix A_i which acts only on the local variables and an *interface* matrix X_i which acts only on the external interface variables. These external interface variables must be first received from neighboring processor(s) before a distributed matrix-vector product can be completed. Thus, each local vector of unknowns $x_i (i = 1, \ldots, p)$ is also split into two parts: the sub-vector u_i of interior variables followed by the sub-vector y_i of inter-domain interface variables. The right-hand side b_i is conformally split into the sub-vectors f_i and g_i,

$$x_i = \begin{pmatrix} u_i \\ y_i \end{pmatrix}, \qquad b_i = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \tag{1}$$

The local matrix A_i residing in processor i is block-partitioned according to this splitting, leading to

$$A_i = \left(\begin{array}{c|c} B_i & F_i \\ \hline E_i & C_i \end{array}\right). \tag{2}$$

With this, the equations assigned to processor i can be written as follows:

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \tag{3}$$

The term $E_{ij}y_j$ is the contribution to the local equations from the neighboring sub-domain number j and N_i is the set of sub-domains that are neighbors to sub-domain i. The sum of these contributions, seen on the left side of (3), is the result of multiplying the interface matrix X_i by the external interface variables. It is clear that the result of this product will affect only the inter-domain interface variables as is indicated by the zero in the upper part of the second term on the left-hand side of (3). For practical implementations, the sub-vectors of external interface variables are grouped into one vector called $y_{i,\text{ext}}$ and the notation

$$\sum_{j \in N_i} E_{ij} y_j \equiv X_i y_{i,\text{ext}}$$

will be used to denote the contributions from external variables to the local system (3). In effect, this represents a local ordering of external variables to write these contributions in a compact matrix form. With this notation, the left-hand side of (3) becomes

$$w_i = A_i x_i + X_i y_{i,\text{ext}}. \tag{4}$$

Note that w_i is also the local part of a global matrix-vector product Ax in which x is a distributed vector which has the local vector components x_i.

2.1. Additive Schwarz preconditioning

Preconditioners for distributed sparse linear systems are best designed from the local data structure described above. Additive Schwarz procedures are the simplest preconditioners available. In the framework of distributed linear systems, see Fig. 1, it can be viewed that there is a "minimal" amount of overlap between the sub-domains. When an additive Schwarz procedure is used as a preconditioner, the local residual vector is updated by a global iterative solver. The resulting local residual vector is then used as the local right-hand side. Of course, an exchange of data must precede the local residual vector update, such that each sub-domain receives from its neighbors the latest values of its external interface variables and sends to the neighbors the latest values of its inter-domain interface variables. Once the latest values of the external interface variables and the local right-hand side are available, the additive Schwarz procedure can be used to find a correction to the local solution. In simple terms, the additive Schwarz preconditioners can be stated as follows:

Algorithm 2.1 (Additive Schwarz with minimum overlap).

1. Update local residual $r_i = (b - Ax)_i$.
2. Solve $A_i \delta_i = r_i$.
3. Exchange δ_i among neighboring sub-domains.
4. Update local solution $x_i = x_i + \delta_i$.

This loop is executed on each processor simultaneously. Exchange of information takes place in Line 1, where the (global) residual is updated. Note that the residual is "updated" in that only the y-part of the right-hand side is changed. The local systems $A_i \delta_i = r_i$ can be solved in three ways: (1) by a (sparse) direct solver, (2) by using a standard preconditioned Krylov solver, or (3) by performing a forward–backward solution associated with an accurate ILU (e.g., ILUT) preconditioner. Experiments show that option (3) or option (2) with only a very small number of inner steps (e.g., 5) is quite effective. In particular, a multilevel ILU type procedure could be used to solve $A_i \delta_i = r_i$ approximately [3,4,21,28,29].

2.2. Schur complement techniques

Schur complement techniques refer to methods which iterate on the inter-domain interface unknowns only, implicitly using interior unknowns as intermediate variables. These techniques form the basis of several preconditioners that will be used in Section 4. Schur complement systems are derived by eliminating the variables u_i from (3). Extracting from the first equation $u_i = B_i^{-1}(f_i - F_i y_i)$ yields, upon substitution in the second equation

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g_i', \tag{5}$$

where S_i is the "local" Schur complement

$$S_i = C_i - E_i B_i^{-1} F_i. \tag{6}$$

Eq. (5) for all sub-domains i ($i = 1, \ldots, p$) constitute a global system of equations involving only the inter-domain interface unknown vectors y_i. This global reduced system has a natural block structure related to the inter-domain interface points in each sub-domain:

$$\begin{pmatrix} S_1 & E_{12} & \ldots & E_{1p} \\ E_{21} & S_2 & \ldots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p-1,2} & \ldots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g_1' \\ g_2' \\ \vdots \\ g_p' \end{pmatrix}. \tag{7}$$

The diagonal blocks in this system, the matrices S_i, are dense in general. The off-diagonal blocks E_{ij}, which are identical with those involved in (3), are sparse.

The system (7) can be written as $Sy = g'$, where y consists of all inter-domain interface variables $y_1, y_2, \ldots, y_p$ stacked into a long vector. The matrix S is the "global" Schur complement matrix. One can exploit approximate solvers (see, e.g., [20]) for the reduced system (7) to develop preconditioners for the original (global) distributed system. Once the global Schur complement system (7) is (approximately) solved, *each* processor will compute the u-part of the solution vector (see (1)) by solving the system $B_i u_i = f_i - E_i y_i$ obtained by substitution from (3). In summary, a Schur complement iteration may be expressed by the following algorithm:

Algorithm 2.2 (Schur complement iteration).

1. Forward: compute local right-hand sides $g_i' = g_i - E_i B_i^{-1} f_i$.
2. Solve global Schur complement system $Sy = g'$.
3. Backward: substitute to obtain u_i, i.e., solve $B_i u_i = f_i - E_i y_i$.

For convenience, (5) is rewritten as a preconditioned system with the diagonal blocks:

$$y_i + S_i^{-1} \sum_{j \in N_i} E_{ij} y_j = S_i^{-1} [g_i - E_i B_i^{-1} f_i]. \tag{8}$$

This can be viewed as a block-Jacobi preconditioned version of the Schur complement system (7). The system (8), which couples local and external unknowns, can be solved by a GMRES-like accelerator, requiring a solve with S_i at each step (see, e.g., [24]).

3. Parallel algebraic recursive multilevel solver

Multilevel Schur complement techniques available in pARMS [18] are based on techniques which exploit block independent sets, such as those described in [27] for the sequential ARMS preconditioner.

3.1. ARMS as the underpinning for preconditioners in pARMS

The sequential version of ARMS is rooted in Multilevel ILU type techniques which exploit *independent sets*. An independent set is a set of unknowns that are not coupled to each other. Such orderings transform the original linear system into a system of the form

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} u \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}, \tag{9}$$

where B is a diagonal matrix. Such independent sets have been extensively used in devising both direct and iterative solution methods for sparse linear systems, see, for example, [11,13,14,21].

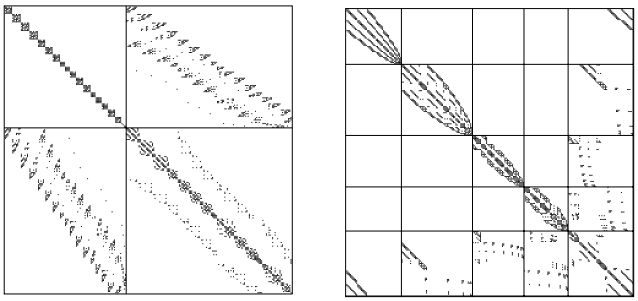
The concept of independent sets has been generalized to *blocks* or *groups* [28,29]. A group-independent set is a collection of subsets (blocks) of unknowns such that there is no coupling between unknowns of any two different groups (blocks) [28]. Unknowns within the same group (block) may be coupled. When the unknowns of the group-independent sets are labeled first, the matrix A will have the block structure (9) in which the B block is block diagonal instead of diagonal. An illustration of a matrix permuted with two different group-independent set orderings is given in Fig. 3.

In ARMS, the following block factorization is computed 'approximately'

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \approx \begin{pmatrix} L & 0 \\ EU^{-1} & I \end{pmatrix} \begin{pmatrix} U & L^{-1}F \\ 0 & A_1 \end{pmatrix}, \tag{10}$$

where $LU \approx B$ and $A_1 \approx C - (EU^{-1})(L^{-1}F)$.

Conceptually, the ARMS factorization algorithm is quite simple to describe because of its recursive nature.



(a) (b)

Fig. 3. Group-independent set reorderings of a 9-point matrix: (a) small groups (fine-grain) and (b) large groups (coarse-grain).

In a nutshell, the ARMS procedure consists of essentially two steps: first, obtain a group-independent set and reorder the matrix in the form (9); second, obtain an ILU factorization $B \approx LU$ for B and approximations to the matrices $L^{-1}F$, EU^{-1}, and A_1. The process is repeated recursively on the matrix A_1, which may now be renamed A, until a selected number of levels is reached. At the last level, a simple ILUT factorization, possibly with pivoting, or an approximate inverse method can be applied. Fig. 4 sketches the ARMS procedure, in which darker shaded areas represent Schur complements that are formed consecutively.

The consecutive Schur complement matrices A_1 remain sparse but will get denser as the number of levels increases, so small elements are dropped in the block factorization to maintain sparsity. The matrices EU^{-1}, $L^{-1}F$ (or their approximations) need not be saved. They are needed only to compute an approximation to A_1. Once this is accomplished, they are discarded to save storage. Subsequent operations with $L^{-1}F$ and EU^{-1} are performed using U, L and the blocks E and F.

Recursive multilevel strategies of various types can be defined. Essentially, a preconditioning step amounts to an approximate solve with the factorization (9). Such a solve consists of three steps. The first step will solve with the first matrix on the right-hand side of (10). This will yield a new right-hand side for the second part of (9), which is $g_1 = g - EU^{-1}f$. Then the resulting Schur complement system $A_1 y = g_1$ is solved (iteratively). Finally, a back-substitution step is performed to obtain the variable $u = U^{-1}[f - L^{-1}Fy]$. It is not specified how the Schur complement system $A_1 y = g_1$ is to be solved—and this provides a source of many possible variations. Recursivity can clearly be exploited: if another level in the ARMS factorization is available, we can repeat the process and *descend* to the next level, solve (recursively), and *ascend* back to the current level. When the last level is reached, one can solve the system by an ILUT-preconditioned GMRES iteration [22] or a simple solve with the ILUT factors (without iteration).

Now let us apply a full-fledged ARMS procedure as a local solver in the construction of a global Schur

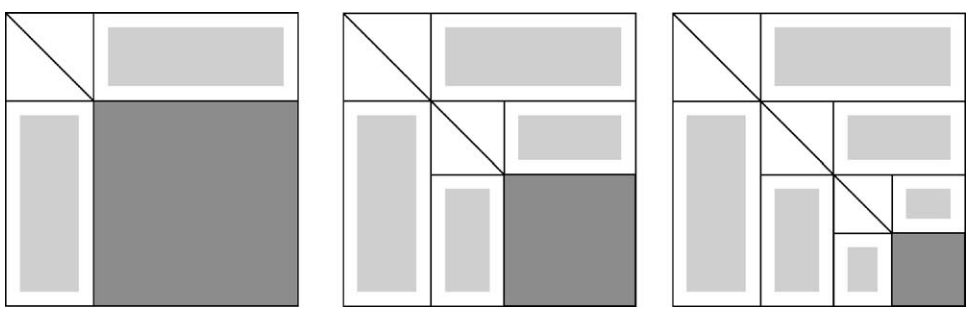


Fig. 4. A step of the ARMS procedure to form consecutive Schur complements.

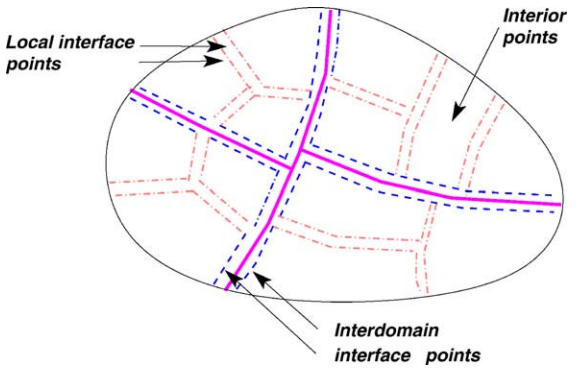


Fig. 5. Classification of unknowns when ARMS is used as a local solver in the pARMS setting.

complement preconditioner. For simplicity, consider a one-level ARMS acting locally in each sub-domain in the pARMS setting. In Fig. 5, solid thick lines determine the boundaries of sub-domains. The local group-independent sets are constructed only on those local unknowns that are decoupled from external unknowns by producing local separators (dashed-dotted lines in Fig. 5). The inter-domain interface unknowns (dashed lines in Fig. 5) are a priori assigned to the Schur complement by the ARMS procedure. Thus in each sub-domain, the local Schur complement matrix A_1, produced in the first level of ARMS reduction, will act on the inter-domain interface variables plus the separator variables, termed the *local interface* variables. In other words, the Schur complement variables $y_i, i = 1, \ldots p$ (see (7)), are the union of the inter-domain interface variables and the variables separating the group-independent sets. We denote by *expanded Schur complement* the system involving the matrix A_1 that acts on the inter-domain and local interface unknowns. Upon solving the global system for all the variables y, the interior variables are obtained without communication by back-substitution in ARMS within each processor.

The paper [27] describes a scalar version of ARMS. The implementation, written in C, is fully recursive and makes efficient use of storage. The paper also presents a number of comparisons with other existing solution methods, both iterative and direct. ARMS is often far superior to ILUT, which is often used as a benchmark for similar comparisons. Its performance is also often better than a standard direct solution package such as SuperLU [9], specifically for large matrices.

3.2. Diagonal shifting in distributed preconditioning

Extremely ill-conditioned linear systems are difficult to solve by iterative methods. A possible source of difficulty is due to the ineffective preconditioning of such systems. The preconditioner may become unstable (i.e., has large norm of its inverse). To stabilize the preconditioner, a common technique is to shift the matrix A by a scalar and use this shifted matrix $A + \alpha I$ during preconditioning, see, e.g., [19]. Because the matrix is shifted, its preconditioner might be a rather accurate approximation of $A + \alpha I$. It is also more likely to be stable. However, for large shift values, the preconditioner might not represent accurately the original matrix A. So the choice of the shift value is important and leads to a trade-off between accuracy and stability of the preconditioner. We have considered this trade-off in [12,25]. In [7], a strong correlation between stability of the preconditioner and the size of $\mathcal{E} = \log(\|(LU)^{-1}\|_{\inf})$ is shown and is suggested as a practical means of evaluating the quality of a preconditioner. We can inexpensively compute $\mathcal{E}_\alpha = \log(\|(LU)^{-1}e\|_1)$, where e is a vector of all ones and LU is an incomplete LU factors of $A + \alpha I$. The estimate $\mathcal{E}_\alpha$ can be used in choosing a shift value: if this estimate is large, then we increase shift value. Before choosing a shift value α, we first scale the coefficient matrix so that all the entries in the matrix have new values that are no larger than one. The scaling process is done by two steps. In the first step, we go through the rows of the matrix one by one. Each row is scaled by dividing all its entries by the 2-norm of the row. In the second step, we go through the columns of the matrix and scale each column in a similar way. When the scaling process is finished, we may simply choose, for example, $\alpha = 1$ and add it to the main diagonal. This will ensure that the new matrix is diagonally dominant, which may not, however, accurately represent the original one. In practice, we can choose an α value between 0 and 1. Once the α value is chosen, we need to recompute (adjust) the preconditioner. Note that efficient techniques for updating a preconditioner when a new shift value is provided is beyond the scope of this paper. One such technique has been outlined in [8].

In pARMS, we have adapted a shifting technique for a distributed representation of linear system. Specifically, we perform the shifting and norm $\mathcal{E}_\alpha$ calculation in each processor independently. Thus, each processor i can have a different shift value depending on the magnitude of its $\mathcal{E}_\alpha^i$. Such an implementation is motivated by the observation that shifting is especially important for diagonally nondominant rows, which can be distinguished among other rows by a local procedure. In each processor, the choice of shift value is described by the following pseudo-code:

Algorithm 3.1 (Matrix shifting).

1. Select initial shift $\alpha \geq 0 : B = A + \alpha I$.
2. Compute parallel preconditioner M for B.
3. Calculate local $\mathcal{E}_\alpha$.
4. If $\mathcal{E}_\alpha$ is large,
   4.1. Increase shift α;
   4.2. Adjust preconditioner.

Note that in Line 4.2 of Algorithm 3.1, depending on the type of preconditioner, the adjustment operation may be either local or global. For example, additive Schwarz type preconditioners may perform adjustments independently per processor, whereas all the processors may need to participate in the adjustment of a Schur complement preconditioner. In addition, Lines 3–6 may need to be repeated several times.

4. Numerical experiments

In this section we describe a few realistic applications, which give rise to large irregularly structured linear systems that are challenging for iterative solution methods. Such applications as ultrasound simulation, magnetohydrodynamics, and tire design are considered. The linear systems arising in ultrasound simulation were generated using Diffpack, which is an object-oriented environment for scientific computing, see [10,17]. The magnetohydrodynamics application has been generously provided by Azzeddine Soulaimani and Ridha Touihri from the "Ecole de Technologie Superieure, Université du Québec", and the linear systems arising in tire design have been supplied by John T. Melson of Michelin Americas Research and Development Corporation. All the numerical experiments have been performed on the IBM SP system at the Minnesota Supercomputing Institute. Each computing node (of type Nighthawk) of the IBM SP system has four 222 MHz Power3 processors sharing 4 GB of memory and all the nodes are connected through a high performance switch. We have tested several preconditioning techniques available in our pARMS code when solving the linear systems iteratively. As the criterion for convergence, we have always required that the residual is reduced by a factor of 10^6.

Before describing the applications, we introduce some notation for the sake of convenience and for helping readers to understand the parallel preconditioners to be used.

- `add_ilut` denotes an additive Schwarz procedure described in Section 2.1. The local system is solved approximately either by a given number of GMRES inner iterations preconditioned with ILUT, or by directly using ILUT as an approximate solver.
- `add_ilu(k)` is similar to `add_ilut` but ILU(k) is used as a preconditioner or an approximate solver instead of ILUT.
- `add_arms` is similar to `add_ilut` but ARMS is used as a preconditioner or an approximate solver for local systems.
- `sch_gilu0` denotes a method that is based on approximately solving the expanded Schur complement system with a global ILU(0)-preconditioned GMRES. The ILU(0) preconditioning requires a global order (referred to as a schedule in [15]) to traverse the equations. For this purpose, global multicoloring of processors is a widely used technique. Since the global Schur complement is never assembled or redistributed in our implementation [24], it can inherit the global multicoloring of the sub-domains (one color per sub-domain corresponding to a processor).
- `no_its` is a suffix that can be added to the above preconditioners to indicate that no inner GMRES iterations are used. For the additive Schwarz type preconditioning, this means that ILUT, ILU(k), or ARMS are used as an approximate local solver. For `sch_gilu0`, global ILU(0) is used in the forward–backward solve on the entire (global) Schur complement.

- `sh` is a suffix that can be added to the above preconditioners to indicate that they use shifted original matrix for the preconditioner construction.

4.1. Simulation of 3D nonlinear acoustic fields

The propagation of 3D ultrasonic waves in a nonlinear medium can be modeled by the following system of nonlinear PDEs:

$$\nabla^2\varphi - \frac{1}{c^2}\frac{\partial^2\varphi}{\partial t^2} + \frac{1}{c^2}\frac{\partial}{\partial t}\left[(\nabla\varphi)^2 + \frac{B/A}{2c^2}\left(\frac{\partial\varphi}{\partial t}\right)^2 + b\nabla^2\varphi\right] = 0, \tag{11}$$

$$p - p_0 = \rho_0\frac{\partial\varphi}{\partial t}, \tag{12}$$

where the primary unknowns are the velocity potential φ and pressure p. For the involved parameters, c is the speed of sound, ρ_0 the density, p_0 the initial pressure, b the absorption parameter and B/A the nonlinearity parameter. The above mathematical model is to be supplemented with suitable initial and boundary conditions.

The numerical scheme consists of using finite elements in the spatial discretization and finite differences for the temporal derivatives. At each time level, the discretization of (11) gives rise to a system of nonlinear algebraic equations involving φ from three consecutive time levels. We apply Newton–Raphson iterations for the nonlinear system. We refer to [5] and the references therein for more information on the mathematical model and the numerical solution method. As a particular numerical test case, we use a 3D domain: $(x, y, z) \in [-0.004, 0.004] \times [-0.004, 0.004] \times [0, 0.008]$. On the face of $z = 0$, there is a circular transducer with radius $r = 0.002$, i.e., the pressure p is given within the circle. On the rest of the boundary we use a nonreflective boundary condition.

We consider solving the linear system during the first Newton–Raphson iteration at the first time level. The linear system has 185,193 unknowns and the sparse matrix contains 11,390,625 nonzero entries. Fig. 6 presents the iteration numbers (left) and total solution times, including preconditioner construction (right), needed for solving this linear system. Two preconditioning techniques, `sch_gilu0_no_its` and `add_arms_no_its` have been tested on various numbers of processors.

It is observed that `sch_gilu0_no_its` preconditioning consistently leads to a faster convergence than `add_arms_no_its`. Both methods, however, show almost no increase in iterations when the number of processors is increased. The timing results are slightly better for `sch_gilu0_no_its` preconditioner except for the 16-processor case.

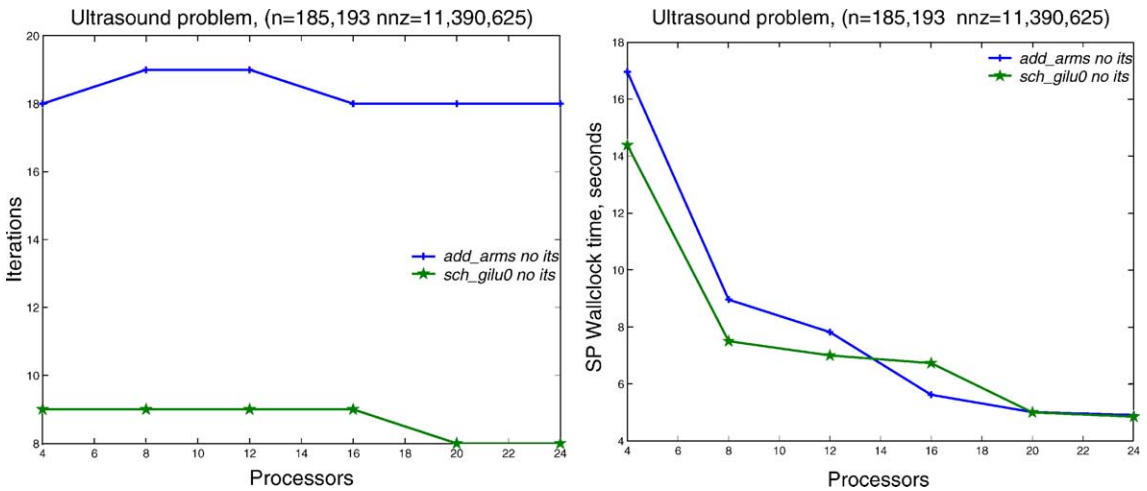


Fig. 6. Outer iterations (left) and timing results (right) for the (fixed-size) ultrasound problem.

4.2. Simulation of magnetohydrodynamic (MHD) flow

In [18], we have described the solution of a rather hard problem which arises from simulating MHD flows. The mathematical model describing the flow consists of the Maxwell equations coupled with the incompressible Navier–Stokes equations. Here, we provide only a brief outline of a sample problem along with its solution and study the solution process when shifting techniques are used. The conservative MHD system is modeled by the Maxwell equations, written as:

$$\frac{\partial \mathbf{B}}{\partial t} - \nabla \times (\mathbf{u} \times \mathbf{B}) + \eta \nabla \times (\nabla \times \mathbf{B}) + \nabla q = 0, \tag{13}$$

$$\nabla \cdot \mathbf{B} = 0, \tag{14}$$

where $\eta, \mathbf{B}, \mathbf{u}$ and q are, respectively, the magnetic diffusivity coefficient, magnetic induction field, velocity field, and the scalar Lagrange multiplier for the magnetic-free divergence constraint. In fully coupled magnetohydrodynamics, this system is solved along with the incompressible Navier–Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} - \nu \nabla^2 \mathbf{u} + \nabla p = \mathbf{f}, \tag{15}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{16}$$

where p, ν and $\mathbf{f}$ are, respectively, pressure, kinematic viscosity, and body force. The coupling between the two systems is through the body force $\mathbf{f} = (1/\mu)(\nabla \times \mathbf{B}) \times \mathbf{B}$, which represents the Lorentz (Laplace) force due to the interaction between the current density $\mathbf{j} = (1/\mu)(\nabla \times \mathbf{B})$ and the magnetic field, where μ is the magnetic permeability.

It is uncommon to solve the fully coupled problem described by Eqs. (13)–(16) along with their coupling via the body forces, because this usually requires an excessive amount of memory. Instead, segregated approaches are often applied which alternatively solve the two coupled problems until a certain convergence criterion is satisfied. For time-dependent problems, these coupling iterations are embedded into the time-stepping procedure. For a few details on this problem, its discretization, and the segregated solution procedure, we refer to [32].

Here, we only consider solving the linear systems arising from the Maxwell equations. In order to do this, a pre-set periodic induction field $\mathbf{u}$ is used in Maxwell's equation (13). The physical region is the 3D unit cube $[-1, 1]^3$ and the discretization uses a Galerkin-least-squares discretization. The magnetic diffusivity coefficient is $\eta = 1$. The sparse matrix of the resulting linear system (denoted by MHD1) has $n = 485{,}597$ unknowns and 24,233,141 nonzero entries. The function q in (13) corresponds to Lagrange multipliers, which arise from imposing the magnetic-free divergence constraint. Its gradient should be zero at steady-state. Though the actual right-hand side was supplied, we preferred to use an artificially generated one in order to check the accuracy of the process. A random initial guess was taken. Little difference in performance was seen when the actual right-hand and a zero vector initial guess were used instead. For the details on the values of the input parameters see [18].

We observed that all the methods without inner iterations experienced stagnation for the MHD1 problem. Additive Schwarz (`add_arms_no_its`) with or without overlap does not converge for any number of processors while the Schur global ILU(0) (`sch_gilu0_no_its`) stagnates when executed on more than nine processors. On four and nine processors, `sch_gilu0_no_its` converges in 188 and 177 iterations, respectively. On the IBM SP system, this amounts to 2223.43 and 1076.27 s, respectively. This is faster than 2372.44 and 1240.23 s when five inner iterations are applied and the number of outer iterations decreases to 119 and 109 on four and nine processors, respectively. Fig. 7, which presents the outer iteration numbers (left) and the timing results (right) for `sch_gilu0` using five inner iterations, indicates that even a few iterations on the global Schur complement lead to convergence for all the tested processor numbers. Using more inner iterations, although beneficial for the convergence rate, may become prohibitively expensive in terms of execution time. This positive effect can be explained by the fact that the Schur complement system is computed with good accuracy. Fig. 7 also shows the usage of the shift value $\alpha = 0.1$ in the `sch_gilu0_sh` preconditioner construction. For this problem, shifting does not help convergence and results in larger numbers of outer iterations. Since a good convergence rate

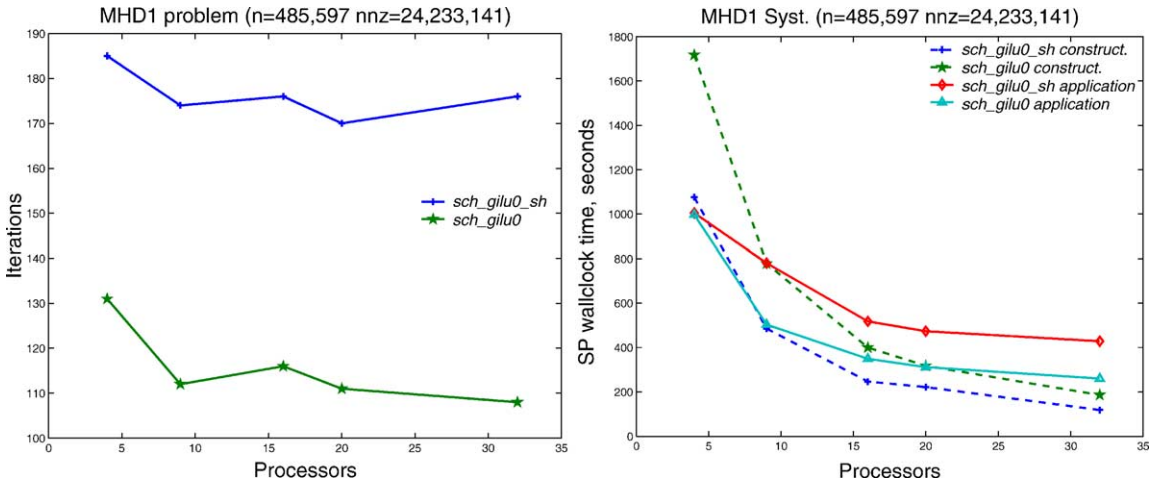


Fig. 7. Outer iteration numbers (left) and timing results (right) for the MHD1 problem with and without diagonal shifting.

is achieved without shifting of the original matrix, the shift value applied in `sch_gilu0_sh` may be too large and the resulting preconditioner may not be a good approximation of the original matrix. The number of nonzeros in `sch_gilu0_sh`, however, is smaller than in `sch_gilu0`. Therefore, the construction of `sch_gilu0_sh` is always cheaper, and `sch_gilu0_sh` appears to be competitive for small processor numbers.

4.3. Linear systems arising in tire design

Tire static equilibrium computation is based on a 3D finite element model with distributed loads (see, e.g., [2]). The computation involves minimizing the potential energy $\Pi(\mathbf{u})$ with respect to the displacement field $\mathbf{u} = (u_1, u_2, u_3)$ subject to nonlinear boundary conditions, which change the symmetry of a tire. The equilibrium equations of the model are obtained by setting the variation $\delta\Pi(\mathbf{u})$ to 0, or equivalently

$$\nabla\Pi(\mathbf{u}) = 0.$$

The Jacobian matrix of the equilibrium equations is obtained by finite difference approximations. The distributed load is scaled by a (loading) parameter λ, and as λ varies the static equilibrium solutions trace out a curve. The difficulty of the finite element problems and concomitant linear systems varies considerably along this equilibrium curve, as well as within the nonlinear iterations to compute a particular point on this curve.

In [31], the problems of varying matrix characteristics are considered. All the problems pose a challenge for iterative methods since the treatment of stationary solutions of rotation makes the systems extremely ill-conditioned during the nonlinear convergence process. It has been observed that an acceptable convergence was achieved *only* when a rather large shift was applied to the diagonal of the scaled matrix. The matrix was scaled to have rows and columns 2-norm unity (see Section 3.2) to stabilize the preconditioner. The size of the shift is very important: while making the preconditioner more stable, large shift values cause the preconditioner to be a poor approximation of the original matrix.

Table 1 shows the results of a few experiments, in which we use preconditioners from pARMS on a sample linear system, medium tire model $\mathcal{M}$, with 49,800 unknowns and, on average, 84 nonzeros per row in the

Table 1
Solution of tire model $\mathcal{M}$ on four processors using different parallel iterative methods (no local iterations are used)

Method	α_{final}	$\max_i \mathcal{E}^i_{\alpha=0}$	`Iter`	`Time`
`add_ilu(2)`	0.1	45	475	175.19
`add_ilut`	0.1	116	476	116.55
`sch_gilu0`	0.2	146	566	99.99

matrix. In distributed preconditioners of pARMS, a shift α is chosen *automatically*: starting with the zero shift, the preconditioner is reconstructed with a new shift (augmented by 0.1) if the estimate $\mathcal{E}_\alpha^i$ of the preconditioner inverse is large (greater than 7). In Table 1, we list the final value of α, the maximum $\mathcal{E}_\alpha^i$ among all the processors when $\alpha = 0$, the number `Iter` of iterations needed for converge, and the preconditioner application time `Time` spent when running on four processors. Metis [16] has been used for the partitioning of the problem among processors.

Note that the `sch_gilu0` preconditioner is more ill-conditioned initially and thus causes two augmentations of the shift value. For `sch_gilu0`, the larger number of outer iterations (566) may be attributed to the resulting preconditioner being much sparser than the other preconditioners tested. However, this difference in the iteration numbers sustain the timing advantage of a sparser preconditioner despite the communication overhead incurred by `sch_gilu0`.

Due to the difficulty of this problem, which is also unpredictably affected by partitioning, the convergence was not observed consistently on any processor numbers. For example, no convergence has been achieved on eight processors for moderate values of α.

5. Concluding remarks

In this paper, we have studied the performance of a recently developed pARMS suite of codes in several realistic applications. For all the problems considered, it is beneficial to use preconditioners based on Schur complement techniques, enhanced by a local multilevel procedure. In addition, a few inner iterations on the global Schur complement secure convergence for a problem arising from a magnetohydrodynamics application.

We have also proposed an implementation of matrix shifting in the framework of distributed linear systems. It allows a shift value to be assigned independently in each sub-domain. An automatic procedure for the shift value selection has also been implemented to stabilize the distributed preconditioner and overcome stagnation.

To conclude, we would like to underline the flexibility of the pARMS framework which allows to select among many available options, by varying the input parameters. This feature is important as it allows users to adapt to the difficulties encountered when solving real-world problems.

Acknowledgements

This work was supported in part by NSF under grants NSF/ACI-0000443 and NSF/INT-0003274, and in part by the Minnesota Supercomputing Institute.

References

[1] S. Balay, W.D. Gropp, L. Curfman McInnes, B.F. Smith, PETSc Users Manual, Technical Report ANL-95/11, Revision 2.1.0, Argonne National Laboratory, 2001.
[2] K.J. Bathe, E.L. Wilson, Numerical Methods in Finite Element Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1976.
[3] E.F.F. Botta, W. Wubs, MRILU: It's the preconditioning that counts, Technical Report W-9703, Department of Mathematics, University of Groningen, The Netherlands, 1997.
[4] E.F.F. Botta, A. van der Ploeg, F.W. Wubs, Nested grids ILU-decomposition (NGILU), J. Comp. Appl. Math. 66 (1996) 515–526.
[5] X. Cai, Å. Ødegård. Parallel simulation of 3D nonlinear acoustic fields on a Linux-cluster, in: Proceedings of the Cluster 2000 Conference.
[6] E. Chow, A. Cleary, R. Falgout, Hypre User's Manual, Version 1.6.0, Technical Report UCRL-MA-137155, Lawrence Livermore National Laboratory, Livermore, CA, 1998.
[7] E. Chow, Y. Saad, Experimental study of ILU preconditioners for indefinite matrices, J. Comp. Appl. Math. 87 (1997) 387–414.
[8] E. Chow, Y. Saad, Approximate inverse preconditioners via sparse–sparse iterations, SIAM J. Sci. Comp. 19 (1998) 995–1023.
[9] J.W. Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li, J.W.H. Liu, A supernodal approach to sparse partial pivoting, SIAM J. Matrix Anal. Appl. 20 (1999) 720–755.
[10] Diffpack World Wide Web home page. http://www.nobjects.com.
[11] J.A. George, J.W.-H. Liu, Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[12] P. Guillaume, Y. Saad, M. Sosonkina, Rational approximation preconditioners for general sparse linear systems, Technical Report umsi-99-209, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1999.
[13] M. Luby, A simple parallel algorithm for the maximum independent set problem, SIAM J. Comp. 14 (4) (1986) 1036–1053.
[14] R. Leuze, Independent set orderings for parallel matrix factorizations by Gaussian elimination, Parallel Comp. 10 (1989) 177–191.

[15] D. Hysom, A. Pothen, A scalable parallel algorithm for incomplete factor preconditioning, Technical Report (preprint), Old-Dominion University, Norfolk, VA, 2000.
[16] G. Karypis, V. Kumar, Metis: Unstructured graph partitioning and sparse matrix ordering system, Technical Report, Department of Computer Science, University of Minnesota, 1995. http://www.cs.umn.edu/~karypis/metis.
[17] H.P. Langtangen, Computational Partial Differential Equations—Numerical Methods and Diffpack Programming, Springer, Berlin, 1999.
[18] Z. Li, Y. Saad, M. Sosonkina, pARMS: A parallel version of the algebraic recursive multilevel solver, Technical Report UMSI-2001-100, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001.
[19] T.A. Manteuffel, An incomplete factorization technique for positive definite linear systems, Math. Comp. 32 (1980) 473–497.
[20] G. Meurant, Domain decomposition methods for partial differential equations on parallel computers, Int. J. Supercomp. Appl. 2 (1988) 5–12.
[21] Y. Saad, ILUM: a multi-elimination ILU preconditioner for general sparse matrices, SIAM J. Sci. Comp. 17 (4) (1996) 830–847.
[22] Y. Saad, Iterative Methods for Sparse Linear Systems, PWS Publishing, New York, 1996.
[23] Y. Saad, A. Malevsky, PSPARSLIB: A portable library of distributed memory sparse iterative solvers, in: V.E. Malyshkin, et al. (Eds.), Proceedings of the Third International Conference on Parallel Computing Technologies (PaCT-95), St. Petersburg, Russia, September 1995.
[24] Y. Saad, M. Sosonkina, Distributed Schur complement techniques for general sparse linear systems, SIAM J. Sci. Comp. 21 (4) (1999) 1337–1356.
[25] Y. Saad, M. Sosonkina, Enhanced preconditioners for large sparse least squares problems, Technical Report umsi-2001-1, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001.
[26] Y. Saad, M. Sosonkina, Solution of distributed sparse linear systems using PSPARSLIB, in: B. Kågström, et al. (Eds.), Applied Parallel Computing, PARA'98, Lecture Notes in Computer Science, No. 1541, Springer, Berlin, 1998, pp. 503–509.
[27] Y. Saad, B. Suchomel, ARMS: An algebraic recursive multilevel solver for general sparse linear systems, Technical Report umsi-99-107-REVIS, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001, revised version of umsi-99-107.
[28] Y. Saad, J. Zhang, BILUM: block versions of multi-elimination and multi-level ILU preconditioner for general sparse linear systems, SIAM J. Sci. Comp. 20 (1999) 2103–2121.
[29] Y. Saad, J. Zhang, BILUTM: a domain-based multi-level block ILUT preconditioner for general sparse matrices, SIAM J. Matrix Anal. Appl. 21 (2000).
[30] B. Smith, P. Bjørstad, W. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, New York, 1996.
[31] M. Sosonkina, J.T. Melson, Y. Saad, L.T. Watson, Preconditioning strategies for linear systems arising in tire design, Numer. Linear Alg. Appl. 7 (2000) 743–757.
[32] A. Soulaimani, N.B. Salah, Y. Saad, Enhanced GMRES acceleration techniques for some CFD problems, Int. J. CFD 16 (2002) 1–20.

M. Sosonkina has graduated from Virginia Polytechnic Institute and State University in 1997 with a PhD degree in computer science and applications. She is an associate professor at the University of Minnesota. She has been a visiting research scientist at the Minnesota Supercomputing Institute in Minneapolis, Minnesota, in 1998, at Ames Laboratory in Ames, Iowa, in 1999. Her research interests include high-performance computing, parallel numerical algorithms for large-scale linear and nonlinear systems, performance analysis, distributed computing, and networking.

Y. Saad received the "Doctorat d'Etat" from the University of Grenoble (France) in 1983. He joined the University of Minnesota in 1990 as a professor of computer science and a fellow of the Minnesota Supercomputer Institute. He was the head of the Department of Computer Science and Engineering from January 1997 to June 2000. From 1981 to 1990, he held positions at the University of California at Berkeley, Yale, the University of Illinois, and the Research Institute for Advanced Computer Science (RIACS). His current research interests include: numerical linear algebra, sparse matrix computations, iterative methods, parallel computing, numerical methods for eigenvalue problems, and control theory.

X. Cai is an associate professor at the Department of Informatics, University of Oslo. He received his PhD degree in applied and industrial mathematics at University of Oslo in 1998. His main research interests are object-oriented software for partial differential equations and parallel/cluster computing. Since 2002 he has taken 80% leave of absence from his university position and is to work at the newly founded Simula Research Laboratory of Norway for a period of 4 years.